

Bachelor-Thesis

in

Medieninformatik

Konzeption, Implementation und Evaluation von Netzwerkkomponenten für eine auf die Lehre spezialisierte Game-Engine

Referent: Prof. Jirka Dell‘Oro-Friedl

Korreferent: Prof. Dr. Ruxandra Lasowski

Vorgelegt am: 31.8. 2100

Vorgelegt von: Falco Böhnke

250100

Im Großacker, 28

79252, Stegen

falco.boehnke@hs-furtwangen.de

Abstract 100-120wort

Im Rahmen dieser Bachelorarbeit wurden Komponenten für die Netzwerkkommunikation innerhalb des auf Electron basierenden Editors „Fudge“ konzipiert und entwickelt. Dafür werden zuerst die Anforderungen, vorgegeben durch Electron und Fudge selbst, festgelegt.

Im Anschluss daran wird erläutert, welche Varianten von Netzwerkkommunikation in heutigen Videospielen üblich sind, welche Protokolle für die Datenübertragung verfügbar sind und welche existierenden Webtechnologien verfügbar sind um mithilfe der jeweiligen Protokolle zu kommunizieren.

Nach Auswahl der zu verwendenden Technologien wird die Konzeption erläutert und die Komponenten umgesetzt und ihre Funktionsweise mithilfe von UML-Diagrammen und Aktivitäts-Diagrammen dargelegt.

Zum Abschluss wird auf die Zukunftsaussichten und etwaige Möglichkeiten der Weiterentwicklung eingegangen und eine Schlussfolgerung über die Entwicklung von Netzwerkkomponenten für einen didaktischen Spieleeditor gezogen.

In the scope of this work network-components for the game-engine and game-editor “Fudge”, which is based on Electron, have been designed and implemented. To begin with, the requirements given by Electron and Fudge are set. Following that is an explanation on how modern videogames communicate via network, which structures and network protocols they use to transmit data, and which Web technologies meet the requirements of Fudge and Electron and allow network communication. After a choice of Web technology is made, the design itself is elaborated on and the structure and functionality of the components are demonstrated using Unified-Modeling-Language-Diagrams and Activity-Diagrams.

In the conclusion there will be a discussion about future development possibilities, possible expansion plans to the components themselves and the upshot is discussed.

Inhaltsverzeichnis

[Abstract 100-120wort I](#_Toc17407855)

[Inhaltsverzeichnis III](#_Toc17407856)

[Abbildungsverzeichnis V](#_Toc17407857)

[Abkürzungsverzeichnis VII](#_Toc17407858)

[1 Einleitung 1](#_Toc17407859)

[2 Rahmenbedingungen – Theoretische Grundlagen 4](#_Toc17407860)

[2.1 Fudge 4](#_Toc17407861)

[2.2 JavaScript und TypeScript 5](#_Toc17407862)

[2.3 Node.js 7](#_Toc17407863)

[2.4 Electron 8](#_Toc17407864)

[2.5 Analyse gebräuchlicher Netzwerkprotokolle 8](#_Toc17407865)

[2.5.1 ISO/OSI-Modell 8](#_Toc17407866)

[2.5.2 Transmission Control Protocol/Internet Protocol 10](#_Toc17407867)

[2.5.3 User Datagram Protocol 12](#_Toc17407868)

[2.5.4 Evaluation UDP und TCP 14](#_Toc17407869)

[2.6 Netzwerkstrukturen in digitalen Spielen 15](#_Toc17407870)

[2.6.1 Clientserver 15](#_Toc17407871)

[2.6.2 Peer To Peer – Meshverbindungen 15](#_Toc17407872)

[2.6.3 Unechtes „Peer to Peer” 16](#_Toc17407873)

[3 Methodik 18](#_Toc17407874)

[Auswahl der kompatiblen Technologien 18](#_Toc17407875)

[Darstellung als grafik zeitstrahl wo inkompatible sachen dann runterfallen sozusagen oder eher plus minus rot grün tabelle 18](#_Toc17407876)

[UDP Kommunikation 18](#_Toc17407877)

[Entwicklung 18](#_Toc17407878)

[Komponenten 18](#_Toc17407879)

[Komp 1 18](#_Toc17407880)

[Komp 2 18](#_Toc17407881)

[Komp 3 18](#_Toc17407882)

[Komp 4 18](#_Toc17407883)

[4. Praktische Anwendung 18](#_Toc17407884)

[Neue Nachrichten abfangen 18](#_Toc17407885)

[Clientenlogik einbauen 18](#_Toc17407886)

[Authoritative Commands strukturieren 18](#_Toc17407887)

[Ids für netzwerkobjete vergeben 18](#_Toc17407888)

[5. Diskussion und Ausblick 18](#_Toc17407889)

Abbildungsverzeichnis

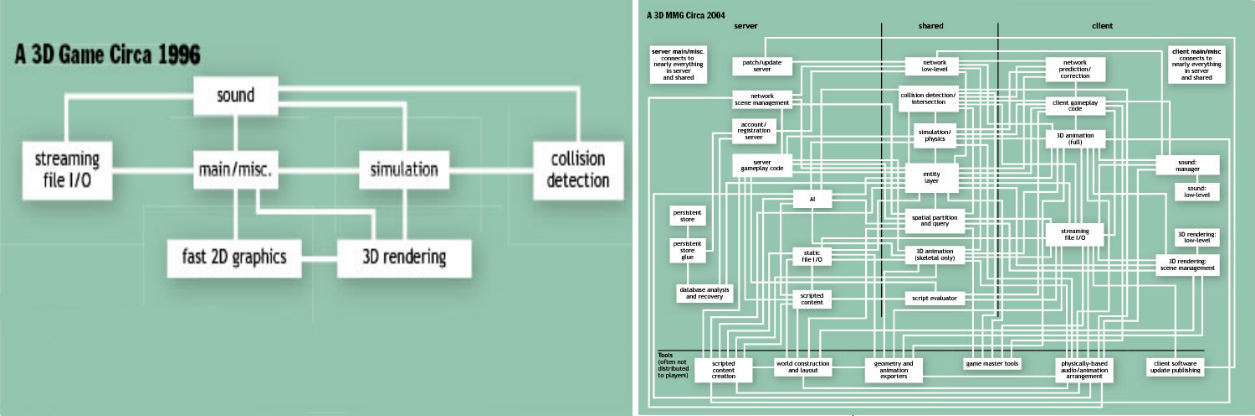
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Abkürzungsverzeichnis

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

1 Einleitung

Die Videospielindustrie ist in den letzten Jahren massiv gewachsen. Allein der Markt für Videospiele in den Niederlanden wuchs von 518 Millionen Euro im Jahr 2012 auf 1.65 Milliarden Euro im Jahr 2019 an, mit vergleichbaren Tendenzen, mit Wachstumszahlen von 11% über die letzten 10 Jahre, weltweit (Wijman 2018) (Kazimier 2017). Der starke Anstieg der Umsatzzahlen hat zu einem gesteigerten Konkurrenzkampf innerhalb des Videospielmarktes geführt. Innovationen sind unerlässlich geworden und so sehen sich Entwickler mit immer komplexeren Projektanforderungen konfrontiert (siehe Abbildung 1). Insbesondere im Bereich der Netzwerkkommunikation sind die Anforderung an hoch performante Lösungen gestiegen.

Abb. 1: Vergleich möglicher, benötigter Komponenten in digitalen Spielen, 1996 und 2004

Quelle: In Anlehnung an Blow, 2004, S. keine ahnung

Spiel-Engines, Entwicklungsumgebungen, die für die Produktion von Videospielen optimiert sind, finden daher verstärkt Einsatz um den Entwicklern die Arbeit zu erleichtern und komplexe Komponenten verwendungsfertig bereit zu stellen. Diese fertigen Komponenten sollen durch Automation und Abstraktion fundamentale Arbeiten übernehmen und so Kapazitäten für Optimierung und Kreativität freisetzen. Doch die Undurchsichtigkeit der abstrahierten Systeme und die oftmals komplizierte Struktur der Spiel-Engines macht es Entwicklern in Ausbildung schwer die grundlegende Funktionsweise der einzelnen Komponenten eines Videospiels zu erfassen und zu erlernen. So sind Entwickler gezwungen statt grundlegender Konzepte die Funktionsweise ihrer ausgewählten Entwicklungsumgebung zu lernen. Ist dann ein Wechsel auf eine andere Entwicklungsumgebung notwendig, müssen Entwickler sich erneut an die Umgebungsentwicklung anpassen, wobei dort ähnliche Komponenten mitunter andere Funktionsweisen haben. Besonders der Mangel einer von Grund auf für die Lehre konzipierten Game-Engine verschärft diese Problematik weiter.

Um diese Lücke zu füllen wurde das Projekt ‚Fudge‘ von Prof. Jirka Dell‘Oro-Friedl ins Leben gerufen. Fudge ist eine Game-Engine und ein Editor, der die Strukturen und Prozesse eines Videospiels offen legt und Entwickler dennoch mit grundlegenden Funktionalitäten versorgt.

In dieser Arbeit wurden Netzwerkkomponenten entwickelt, die Entwicklern in Fudge die Möglichkeit geben vernetzte Spiele zu entwickeln, ohne die notwendigen Komponenten von Grund auf selbst zu schreiben. Dazu wird zuerst der Entwicklungsrahmen mit Fudge und Electron dargelegt. Anschließend wurden die in modernen Spielen üblichsten Methoden zur Kommunikation zwischen Spielinstanzen ermittelt und kurz erläutert. Dazu gehört ebenfalls ein kurzer Exkurs in modernen Netzwerkprotokollen die in der Netzwerkkommunikation üblich sind und daher relevant für die Umsetzung der Netzwerkkommunikation in Fudge.

Aus diesen Informationen erschließen sich die Anforderungen an die Netzwerkkomponenten und die Aufgaben die sie erfüllen können müssen, um für die Entwicklung von Spielen in Fudge sinnvoll verwendet werden zu können.

Unter Berücksichtigung der Rahmenbedingungen und Anforderungen werden dann die möglichen Technologien ermittelt, die bei der Entwicklung verwendet werden können. Diese werden evaluiert und die passenden werden im weiteren für die Entwicklung verwendet

Darauf folgend wird die Umsetzung der einzelnen notwendigen Netzwerkkomponenten dargelegt und ihr Aufbaue mithilfe von Unified Modeling Language Diagrammen dargestellt, sowie ihre Funktionsweise und Ablauf beispielhaft mithilfe von Aktivitätsdiagrammen dargestellt.

Abschließend werden die Ergebnisse der Evaluation und Entwicklung zusammengefasst und ein Fingerzeig für eine mögliche Weiterentwicklung der Komponenten gegeben.

2 Rahmenbedingungen – Theoretische Grundlagen

In diesem Kapitel werden die Rahmenbedingungen für die Entwicklung der Netzwerkkomponenten dargelegt, welche Einschränkungen und Besonderheiten existieren und welche besonderen Anforderungen durch Fudge und Electron entstehen.

2.1 Fudge

Fudge ist eine open Source die Game-Engine und Spieleditor für die Entwicklung von zwei dimensionalen und drei dimensionalen Sipelen kombiniert und besonders augenmerk auf didaktisch sinnvolle Strukturen sowie menschenlesbaren Quellcode legt. Der Name gründet sich dabei aus dem vollen Titel der Software: „Furtwangen University Didactic Game Editor“. Als Game-Engine übernimmt Fudge dabei grundlegende Aufgaben wie die visuelle Repräsentation des Spielverlaufs sowie dem allgemeinen Ablauf, Audiowiedergabe und Animationen. Dazu kommt, auf seiten des Spieleditors, eine einfache Darstellung der Komponenten die in einem gegebenen Spielzustand verwendet werden können, sowie deren Werte. Fudge ähnelt in dieser Hinsicht stark den gebräuchlicheren kommerziellen Game-Engines wie Unity, die Unreal Engine und Frostbtte, die ebenfalls Editor und Engine kombinieren. Gegenüber diesen kommerziellen Produkten ist Fudge allerdings auf die Anwendung in der akademischen Ausbildung spezialisiert und hat dadurch in diesem Bereich klare vorteile. Datenformate sind allgemein gültig und menschenlesbar gestaltet, das bedeutet der Quellcode soll sich durch geschickte Namensgebung und klare Strukturen selbst erklären können. Außerdem wird so die Versionskontrolle vereinfacht, da eine Integration mit Anbietern wie Github oder Gitlab von grundauf eingeplant und möglich ist. Ein weiterer wichtiger Punkt ist, dass Fudge sich auf das prinzip von „Composition over inheritance“(Coi) stützt. Dem CoI-Prinzip folgend werden Komponenten kreeirt die wiederverwendbar sind und aus denen sich andere Klassen zusammensetzen lassen. Außerdem reduziert das CoI-Prinzip die Notwendigkeit Gemeinsamkeiten zwischen Klassen zu finden, um sie in sinnvolle Familienbäume zusammenzufassen.

Fudge basiert in seiner grundlegenden Struktur auf Webtechnologien, namentlich HTML, CSS, JavaScript und TypeScript, NodeJS und Electron in Kombination mit der Browserumgebung Chromium. Diese erlauben in Kombination mit Elektron für einen auf den geläufigsten Desktopgeräten verwendbaren Editor, dessen gepackte Endprodukte crossplattformkompatibel sind, das bedeutet auf verschiedenen Betriebssystemen, Geräten und sogar Browsern nutzbar sind.

Besonderes Augenmerk liegt im Rahmen dieser Bachelorarbeit auf TypeScript, NodeJS und Electron, da Entscheidungen über Kompatibilität mit Einschränkungen und Möglichkeiten getroffen werden müssen, die von diesen Technologien ausgehen. Im Folgenden wird kurz erläutert, was die einzelnen Technologien sind und an Möglichkeiten bieten, und inwiefern sie die Entwicklung der Netzwerkkomponenten für Fudge beeinflussen.

2.2 JavaScript und TypeScript

JavaScript ist, wie der Name impliziert, eine Skriptsprache und eine der Kerntechnologien auf denen das World Wide Web aufbaut. Eine Skriptsprache ist dabei explizit designed um auf ein existierendes System oder eine vorhanden Entität aufzubauen. Als solches folgt JavaScript dem international anerkannten ECMAScript Standard. Dieser Standard besagt welche Typen, Werte, Objekte, Eigenschaften, Funktionen und Programm Syntax sowie Semantic von JavaScript angeboten und unterstützt werden müssen

JavaScript ist zudem eine interpretierte, keine kompilierte Programmiersprache Kompilation bedeutet, dass Quellcode von einer Software, dem Compiler, in maschinencode übersetzt wird. Der Compiler liefert als Resultat ein Programm in Maschinencode zurück. Ein Interpreter dagegen liefert das Ergebnis eines Quellcodes zurück, also das Ergebnis einer Berechnung. Allerdings muss beachtet werden dass die Grenzen zwischen interpretierten und kompilierten Sprachen immer mehr verwischen, da mit der V8 JavaScript Engine auch Javascript zu Maschinencode kompiliert werden kann. Dies bringt Geschwindigkeitsvorteile mit sich.

Da JavaScript in jedem modernen Browser nutzbar ist können angehende Entwickler schnell erste Erfolge erzielen. Zudem lassen sich komplexe JavaScript Abläufe Stück für Stück programmieren und im Browser auf ihre funktionalität testen. Dies erleichtert den Einstieg erheblich und senkt so die Lernkurve stark ab.

Außerdem existieren viele Fallstricke anderer Programmiersprachen in JS nicht, unter anderem Einschränkungen durch Typisierung und komplexe Sturkturen durch multiple Vererbung, die Erfahrung und solide Planung voraussetzen.

Ein weiterer Vorteil, der JavaScript als Grundbaustein des World Wide Web zementierte, ist die Codeausführung auf Seiten des Clienten. Dabei liefert ein Server lediglich den Quellcode an dne Clienten, das Programm wird jedoch im Browser des Clienten ausgeführt.So lassen sich Daten bis zu einem gewissen Grad validieren, ohne dafür Serverressourcen verwenden zu müssen. Dies geschieht auf Kosten der Sicherheit. Browser injezierte JavaScript Programme können Daten auslesen und verschicken und sogar Schäden an den ausführenden Geräten selbst verursachen. Außerdem erschwert die fehlende Typisierung und der nicht existente Debugger, Software die Fehler im quellcode erkennt und ausliest, die Wartung ungemein und führt zu Laufzeitfehlern, die nur durch zeitintensive Fehlersuche behoben werden können. Dazu kommt, dass die Verwendung von Objekt-orientierten Prinzipien durch die fehlende Typisierung und ausschließlich einfache Vererbung viel Kreativität erfordert.

Dennoch erfreut sich JS sehr großer Beliebtheit und ist heute, 2019, der defacto Standard für moderne Browser, unterstützt von Google, Facebook und Mozilla.

Einige der Nachteile von JavaScript lassen sich jedoch mithilfe von Typescript beheben und umgehen. TypeScript ist eine von Microsoft entwickeltes, typisiertes Superset von JavaScript. Quellcode wird in TypeScript geschrieben und der Typescript eigene Compiler wandelt den Code in funktionierendes JavaScript um. Durch die nun vorhandene Typisierung und einen TypeScript eigenen Debugger sind Fehler jedoch sbereits im Quellcode leicht ersichtlich. Klassen, Objekte und ihre Funktionen lassen sich deklasieren und festlegen; sind eindeutig definiert und können während der Laufzeit nicht ungewollt verändert oder überschrieben werden. So lassen sich objektoriente Programmierprinzipien erfolgreich und problemlos anwenden, der Quellcode ist dadurch übersichtlich strukturiert, die Wartbarkeit ist vereinfacht und der Quellcode lässt sich mit allgemeingültigen Modellen, wie der Unified Modelling Language, darstellen und dokumentieren. Die Beliebtheit von TypeScript steigt ob dieser Vorteile stetig an und ersetzt mehr und mehr Entwicklung mit reinem JavaScript.

Eine Einschränkung die JavaScript der Entwicklung von Netzwerkkomponenten für eine browserähnliche Umgebung auferlegt, sind die Sicherheitsbestimmungen für die Kommunikation aus dem Browser heraus. So ist JavaScript im Browser weder in der Lage direkt über das Transmission Control Protocol oder über das User Datagram Protocol Daten zu versenden. Dies ist von Mozilla ebenso wie von Google bestimmt worden um die Ausführung von bösartigem Code zu verhindern, der Netzwerkverbindungen öffnet und beliebig Daten versenden könnte.

Für die Entwicklung in Fudge wird ausschließlich TypeScript verwendet, da der resultierende, noch nicht kompilierte Quellcode sich menschenlesbar gestalten lässt, ein Debugger zur Verfügung steht und sich der Code durch die Möglichkeiten der Objektorientierten Programmierung in modulare Komponenten aufspalten lässt. JavaScript, entstanden aus TypeScript oder direkt geschrieben, ist nicht in der Lage ist Datenpakete aus einem Browser heraus zu verschicken. Warum dies relevant ist wird im Laufe dieses Kapitels geklärt und es werden Technologien evaluiert, die diese Limitierung aufheben.

2.3 Node.js

Node.js ist eine Laufzeitumgebung, die alle Komponenten beinhaltet, die für die Ausführung von JavaScript Code notwendig sind. Node.js basiert, wie JavaScript selbst, auf der V8 JavaScript Runtime Engine und wird zu maschinencode übersetzt. Node.js ist durch seine Autarkität geeignet, um in Serverumgebungen zu arbeiten. Die eigenständige Laufzeitumgebung ermöglicht es JavaScript für die direkte Datenübertragung und Verarbeitung zu benutzen.

Außerdem bietet Node.js, im Gegensatz zu reinem JavaScript, dessen Event-Schleife nur auf einem Thread arbeitet, die Möglichkeit asynchron auf mehreren Threads zu arbeiten und so nicht blockierend Anfragen zu verarbeiten. So lassen sich viele Anfragen gleichzeitig statt nacheinander abarbeiten. Dies ist besonders kritisch in einer Serverumgebung da ansonsten jede Anfrage nacheinander einzeln bearbeitet werden muss, und sich so Anfragen gegenseitig blockieren. Durch die asynchrone Arbeitsweise kann Node.js nicht-blockierende Input/Output Requests anbieten.

Node.js erfreut sich dabei ebenso wie JavaScript großer Beliebtheit wodurch viele Bibliotheken und Module zur freien Verfügung stehen. Über den node.js eigenen Package-Verwalter „npm“ können diese problemlos abgerufen, installiert und integriert werden. Dies macht Node.js kompatibel mit den meisten geläufigen Programmiersprachen, APIs und Programmen.

Node.js bietet außerdem grundlegende Komponenten der Netzwerkkommunikation, auf die im späteren eingegangen wird.

2.4 Electron

Electron basiert auf dem Projekt Atom Shell das im Jahr 2013 begann und im Jahr 2015 zu Electron umbenannt wurde. Initial war es als plattformübergreifender Text Editor gedacht, in welchem der Nutzer mit Webtechnologien wie JavaScript CSS und HTML arbeiten kann.

Electron kombiniert die Chromium Rendering Library, die ein open-source Fundament für Google Chrome darstellt, mit Node.js und der V8 JavaScript Engine in einer einzigen Laufzeitumgebung. Dadurch können Programme die in JavaScript oder Node.js geschrieben werden ohne Einschränkungen verwendet werden und neue Programme auf Basis von Node.js, HTML/CSS und JavaScript erstellt werden. Diese Programme sind plattformübergreifend kompatibel und können sowohl für Desktop, Mobile und Browserumgebungen ausgegeben werden. So können Webentwickler auch vollwertige Desktopsoftware und sogar Applikationen für Smartphones erstellen, ohne von ihren bekannten Technologien abweichen zu müssen. Wichtig ist jedoch zu beachten, dass Electron auf einer Browserumgebung aufbaut und dementsprechend Einschränkungen im Bereich des Dateizugriffs aufweist, sowie die browserüblichen Sicherheitssysteme für die Netzwerkkommunikation.

2.5 Analyse gebräuchlicher Netzwerkprotokolle

2.5.1 ISO/OSI-Modell

Beim ISO/OSI Modell (kurz: OSI-Modell) ist ein Produkt der ISO, dem ein Standardisierungswunsch zugrunde liegt. Da die Gesamtheit der Funktionalitäten der Datenkommunikation zu komplex sind um verständlich zu sein, wurden „… gemäß dem Konzept der virtuellen Maschinen mehrere Schichten ausgedacht, um die Materie etwas übersichtlicher zu beschreiben“ (TCPUNDUDPINTERNALS ZITIEREN SEITE 14). Das so entstandene OSI-Modell unterteilt dabei die Funktionalitäten der Datenkommunikation in sieben gestapelte Schichten, von denen jede Schicht mit der darunterliegenden über Schnittstellen kommunizieren kann.

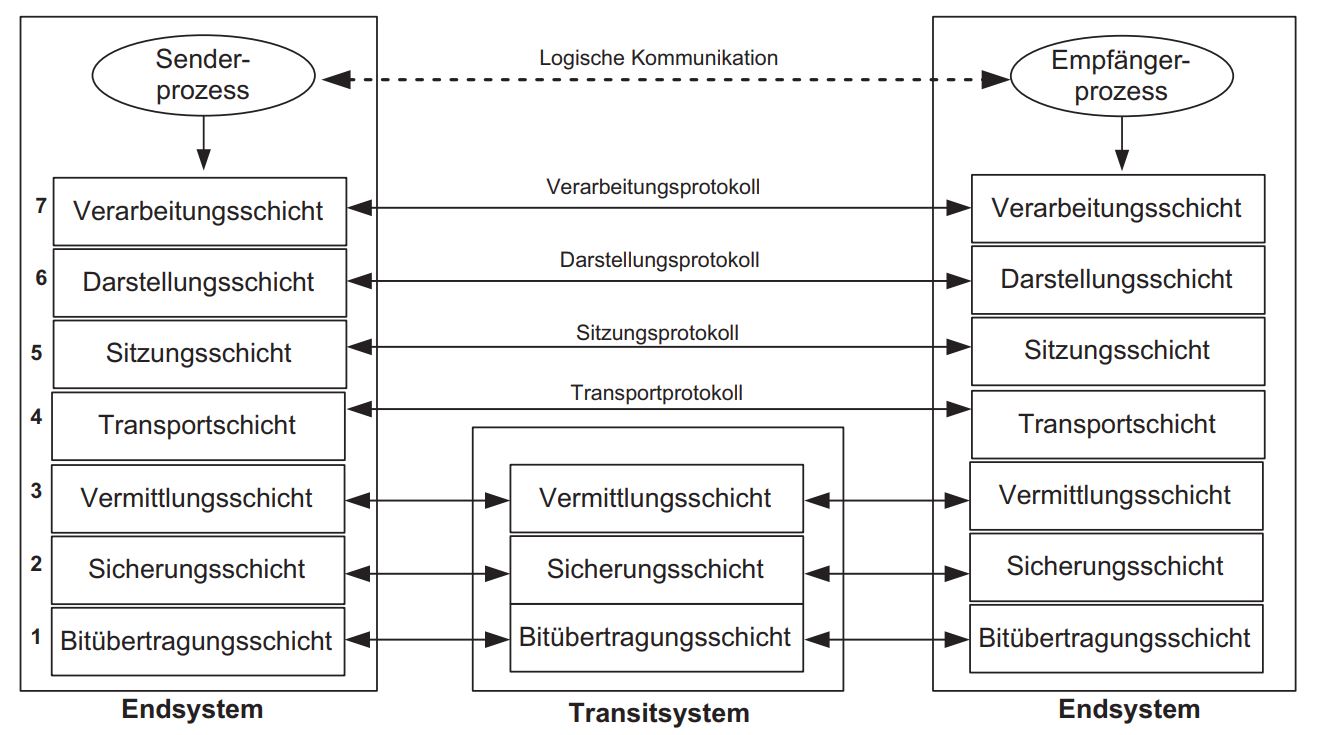


Abb. 2: Das ISO/OSI Modell

Quelle: Mandl, 2017, S. 14

Schichten 1-4 (Siehe Abbildung 2) werden dabei „…gemeinsam als Transportsystem bezeichnet.“ (ZITAT BUCH) Wichtig hierbei ist die Aufgaben der einzelnen Schichten im Transportsystem zu unterscheiden: Schicht 2 und Schicht 3 dienen dazu Verbindungen zwischen Rechnern herzustellen, Schicht 2 ermöglicht direkte Ende-zu-Ende Verbindungen zwischen Rechnern, unter Verwendung von Protokollen wie dem Internet Protocol (IP), Schicht 3 fügt diesem die Möglichkeit hinzu in einem Netzwerk, das heißt über mehrere Knoten, eine Ende-zu-Ende Verbindung zwischen Rechnern herzustellen Für diese Arbeit relevant ist die vierte Schicht, die sogenannte Transportschicht. Diese „… kümmert sich um die Ende-zu-Ende-Kommunikation zwischen zwei Prozessen auf einem oder unterschiedlichen Rechnern.“ (ZITAT) Sie ist also dafür zuständig, dass Software, im Falle von Fudge sind es digitale Spiele oder interaktive Anwendungen, direkt miteinander kommunizieren können. Diese Kommunikation wird über sogenannte Packets bewerkstelligt, die einen Header mit Informationen wie Quelle, Ziel und Status enhalten, und einen Körper, der die Nutzdaten beinhaltet, besitzen. (ZITAT) Der Transportschicht stehen zwei hier relevante Datenprotokolle zur Übertragung und Zuweisung dieser Packets zur Verfügung: das User-Datagram-Protocol und das Transmission-Control-Protocol. Hier ist die Unterscheidung zwischen Transmission Control Protocol (TCP) und Tranmission Control Protocol/Internect Protocol (TCP/IP) wichtig: TCP/IP bezeichnet die gesamte Protokollfamilie die für die Kommunikation über Netzwerke und Internet verwendet wird. Der Name erschließt sich aus den zwei wichtigsten Protokollen, dem TCP und dem IP. Dementsprechend bezeichnet TCP ein spezifisches Protokoll innerhalb der TCP/IP Gruppe. Im weiteren Verlauf der Arbeit wird auf das spezifische Protokoll TCP eingegangen. Im Folgenden wird auf das TCP und das User Datagramm Protocol (UDP) eingegangen.

2.5.2 Transmission Control Protocol/Internet Protocol

Das Internet Protocol ist die erste vom Übertragungsmedium unabhängige Schicht des OSI-Modell. Es erlaubt Computer in logische Einheiten, genannt Subnetze, zu gruppieren und die Computer dann mit eindeutigen Adressen, den IPs anzusprechen. Ein Beispiel für eine solche IP ist der localhost, der eigene Netzwerkanschluss, eines jeden Geräts der üblicherweise unter der IP-Adresse 127.0.0.1 zu finden ist.

Mit IP als zugrundeliegender Schicht im OSI-Modell, hat TCP sich zum heute meistgenutzten Protokoll der Netzwerkkommunikation über das Internet aufgeschwungen. TCP ist ein zuverlässiges, verbindungsorientiertes Protokoll. Zuverlässig im Kontext der Netzwerkkommunikation bedeutet, dass TCP die Ankunft von Packets in der korrekten, nummerierten Reihenfolge garantiert. Sollte ein Packet verloren gehen, beispielsweise durch eine schlechte Internetverbindung, so werden alle Packets zurückgestellt, bis das verlorene Packet nachgesendet und schlussendlich empfangen wurde. Dies macht TCP das Protokoll der Wahl wenn Packets und ihre Daten nicht verändert werden, nicht verloren gehen und nicht dupliziert werden dürfen, und zudem in der richtigen Reihenfolge eintreffen sollen.

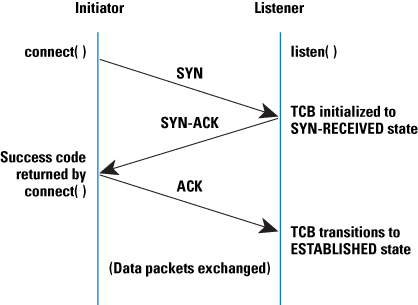
 Es mus zuerst Verbindungsorientiert bedeutet zwingend: eine Verbindung zwischen Clients, per Drei Wege Handshake Verhandlung (siehe Abbildung 3) erstellt werden, ehe Packets versendet werden können. Dies erfordert die Initiierung der Verbindung seitens eines Computers, eine Anerkennung der Verbindungsanfrage seitens des Empfängers und schlussendlich eine Bestätigung des Initiators. Sind diese drei Schritte erfolgreich, so wird eine Verbindung zwischen den Computern geöffnet.

Abb. 3: Dreiwege Handshake Verhandlung zwischen zwei Computern

Außerdem erlaubt TCP dadurch eine „… vollduplex-fähige, bidirektionale virtuelle Verbindung zwischen Anwendungsprozessen“. Dies ermöglicht es allen beteiligten Kommunikationspartnern zu beliebigen Zeitpunkten und sogar zeitgleich Nachrichten zu senden und zu empfangen. (ZITAT)

Für die Verwendung in Login-Servern oder Lobby/Matchmaking-Servern bietet sich TCP an, jedoch kostet der Verbindungsaufbau Zeit und der Datenaustausch erhält eine größere Latenz, Verzögerungen in Übertragung und Interpretation der Packets, durch die ausführlichen Header Daten.

TCP eignet sich dagegen nicht für den Datenaustausch innerhalb von Echtzeitanwendungen. Durch die zuverlässige Verbindung, die TCP erfordert, müssen Packets in der korrekten Reihenfolge eintreffen. Geht ein Packet verloren wird der komplette Datenverkehr eingestellt, bis das verlorene Packet eintrifft oder es durch Nachsenden empfangen wird, dies nennt sich Netzwerkstau. Besonders deutlich wird dies in Echtzeitspielen oder Audioübertragungen: Geht ein Packet in einem Spiel verloren wrid die Verbindung des Clients solange blockiert, bis das Packet eintrifft. In dieser Zeit findet keine Aktualisierung des Spielstandes statt und eine merkliche Verzögerung tritt auf. Ebenso ist dies bei der Übertragung von Audio oder Video der Fall. Bild und Ton werden solange angehalten, bis das Packet eintrifft. So fehlt zwar keine Information, es wird dennoch als eindeutiges Ruckeln empfunden. Dadurch disqualifiziert sich TCP für die Kommunikation zwischen Echtzeitspielen, da keine gute Netzwerkverbindung der Clients gewährleistet werden kann. Allerdings kann TCP gut für den initialen Verbindungsaufbau zu einem Login-Server oder einer Spiel-Lobby verwendet werden, da dort zuverlässige Kommunikation selbst mit Zeitverzögerung wichtig sein kann.

In der Funktionsweise ist TCP vergleichbar mit einer traditionellen Telefonverbindung: Der Anrufer äußert einen Kommunikationswunsch, der Angerufene erkennt diesen Wunsch an indem er den Anruf entgegennimmt und seinen Kommunikationsbereitschaft mit einem Gruß signalisiert. Diese erkennt der Anrufer ebenfalls mit einem Gruß an und die Kommunikation kann beginnen.

2.5.3 User Datagram Protocol

Im Gegensatz zum TCP steht das User Datagram Protocol (UDP). Es ist weder verbindungsorientiert, noch zuverlässig. Es eignet sich besonders für die Übertragung von Packets, deren Verlust nicht systemkritisch ist, da der Empfang der Daten nicht garantiert wird. Gleichzeitig besteht keine Gefahr eine Verbindung bis zum Eintreffen eines Packets zu blockieren. So lässt sich diese Eigenschaft von TCP umgehen. Zudem ist UDP nicht Verbindungsabhängig, die Kommunikation erfolgt direkt über UDP-Kommunikationsendpunkte. Diese bestehen aus einem Tupel das zum einen die IP und zum anderen den UDP Port der Anwendung enthält. Dies erlaubt es mit UDP Broadcastnachrichten zu versenden. Außerdem ist der Header eines UDP Packets (auch: Datagram) wesnetlich leichtgewichtiger, wodurch die Latenz bei Versand und Empfang der Nachrichten reduziert ist. Die Unzuverlässigkeit von UDP erfordert jedoch, dass Entwickler auf verlorene Nachrichten oder Nachrichten die in der falschen Reihenfolge eintreffen Rücksicht nehmen müssen, und eigenständig eine Ordnung vornehmen und beispielsweise UDP-Packets, die vor dem zuletzt eingetroffenen verschickt aber nicht empfangen wurden, verwerfen.-

Das UDP ist vergleichbar mit einem Radiosender: Unabhängig davon ob es Empfänger gibt, können die Informationen verschickt werden. Clients, die ihre Radios auf den Empfang der Funkwellen justieren werden die Nachrichten empfangen, andere nicht. Gehen Teile der Übertragung verloren, so setzt die Übertragung aus und beginnt erneut sobald neue Informationen empfangen werden.

Durch diese Eigenschaften wird die Übertragung von Spielständen oder Tastendrücken in digitalen Spielen heutzutage größtenteils per UDP gehandhabt. Verlorene Packets haben selten kritische Auswirkungen auf die Systeme des Spiels, störendes Ruckeln durch Netzwerkstau werden vermieden und durch die Broadcast-Eigenschaften von UDP lassen sich mehr Clients mit weniger Aufwand als bei TCP ansprechen.

2.5.4 Evaluation UDP und TCP



Abb. 4: Gegenüberstellung der Eigenschaften von UDP und TCP in Anlehnung an Vasundhara Ghose

Quelle: Ghose, keine ahnung, keine ahnung

Wie in Abbildung 4 zu sehen, haben beide Technologien ihre Besonderheiten, die sie in ihrem Nutzen voneinander trennen. Für die Kommunikation in Echtzeitspielen ist dabei UDP als Transportprotokoll zu bevorzugen, da aufgrund der fehlenden Ordnung und Zuverlässigkeit Netzwerkstaus und Aussetzer wegfallen.

TCP dagegen bietet sich besonders für Lobbyserver, Loginserver und Filetransfers an, da die Daten garantiert und in der korrekten Reihenfolge gesendet werden. Dies erspart auch die Notwendigkeit die Packets zusätzlich validieren zu müssen.

2.6 Netzwerkstrukturen in digitalen Spielen

2.6.1 Clientserver

Eine der ältesten und meist genutzten Varianten der Netzwerkkommunikation in Spielen stellt die Client-Server Struktur dar. Dabei gibt es einen dedizierten Server, der als Dreh und Angelpunkt für alle Clientinteraktionen mit dem Spiel und miteinander dient. Der Server arbeitet Anfragen ab, synchronisiert Spielstände, validiert Eingaben der User und leitet relevante Informationen an die Clients weiter. Dies hat sich insbesondere bei rasanten Spielen wie Multiplayer-Online-Shootern wie beispielsweise Apex Legends, Overwatch und Fortnite durchgesetzt, da die Server in einer stabilen Infrastruktur aufgebaut werden und ganztägig betrieben werden können und so stetige und vor allem ruckelfreie Spielerlebnisse bieten können. Latenz und Verbindungsabbrüche sind hierbei üblicherweise vom Client ausgelöst und müssen so seitens der Anbieter nicht gewartet oder anderweitig unterstützt werden. Zudem erlaubt eine Client-Struktur die Kontrolle des Spielerlebnisses und den Ausschluss von Spielern, da die Spiele ausschließlich dann funktionieren, wenn sie Kontakt zu einem der Spieleserver haben. Gleichzeitig erfordert diese Art der Netzwerkkommunikation eine solide Serverinfrastruktur, da ansonsten das Spielerlebnis für alle Spieler gleichermaßen unter Lags, Verzögerungen die durch langsame Übertragung von Daten entstehen, leidet.

Die Client-Server Struktur ist die häufigste Struktur der Netzwerkkommunikation die in digitalen Spielen verwendet wird. Sie bietet, adäquate Serverinfrastruktur vorausgesetzt, die geringste Latenz bei der Verarbeitung von Daten, erlaubt es viele dutzend oder sogar hunderte von Spielern in das Spiel einzubinden und erlaubt den Entwicklern volle Kontrolle über das Spiel, einschließlich der Kontrolle über die Clients die den Service nutzen dürfen und wie lange der Service zur Verfügung steht. Hohe initiale Kosten, Betriebskosten und Wartungskosten seitens der Betreiber, sowie das benötigte Fachwissen um den Spielserver sicher und stabil aufzubauen, erschweren jedoch den Einsatz durch kleinere Studios mit begrenzten Ressourcen.

2.6.2 Peer To Peer – Meshverbindungen

Seltener genutzt wird das Peer-To-Peer Verbindungsverfahren. Hierbei gibt es keinen dedizierten Server. Stattdessen verbinden sich alle Clients miteinander und erzeugen ein sogenanntes Mesh. Spielstände werden zwischen den Clienten synchronisiert und Eingaben gehen von den initiierenden Clienten in das Netzwerk, werden von den empfangenden Clienten verarbeitet und angezeigt. Dies erlaubt es Serverfreie Multiplayerspiele zu erschaffen und so die Kosten für die Bereitstellung und Wartung der Serverinfrastruktur zu umgehen. Besonders bei kleineren Studios und einzelnen Entwicklern kann dies ein entscheidendes Kriterium sein. Peer To Peer Verbindungen sind dafür fehleranfällig, da die Verbindungsqualität aller Clients oftmals von jedem andere Client abhängig ist. So kann eine schlechte Verbindung das Spielerlebnis vieler einschränken, auch wenn dies durch geschickte Planung und entsprechende Kontrollsysteme reduziert werden kann. Außerdem gibt es keine zentrale Validierung der ins Netzwerk gesendeten Daten. Dadurch sind Peer to Peer Meshes anfällig für Dateninjektionen oder andere Betrügereien, die einem bestimmten Spieler massive Vorteile bieten. Zu guter Letzt lassen sich so keine größeren Multiplayerspiele realisieren, da die Organisation eines Peer to Peer Meshes eine Vielzahl von Einzelverbindungen erzeugt. In einem 20-Spieler Mesh muss jeder Client 19 Verbindungen verwalten, wodurch die Hardwareanforderungen stark ansteigen.

Diese Art der Netzwerkkommunikation findet hauptsächlich in kleineren, unabhängig entwickelten digitalen Spielen Anwendungen, deren Spielerzahl häufig im Rahmen des Spieldesigns auf wenige Spieler beschränkt sind. Die Einfachheit der Entwicklung und der vollständige Verzicht darauf eine Serverinfrastruktur bereitstellen zu müssen sind hierbei klare Vorteile dieser Verbindungsmethode.

2.6.3 „Unechtes“ Peer to Peer

Bei unechtem Peer to Peer handelt es sich eigentlich um eine Client-Server Variante. Hierbei werden Clients üblicherweise per Matchmaking oder Lobbys zusammengeführt. Der Matchmaking/Lobbyserver organisiert dann die Verbindungen, indem er einen Client auswählt der als „Host“, also Client und Server in Kombination fungiert und die Rolle des Servers für alle anderen Clients übernimmt. Anschließend verbinden sich alle Clients innerhalb der Lobby mit diesem Server, der kein dedizierter Server sondern ein Peer (übersetzt „Gleichgestellter“) ist. In der Funktionsweise agiert diese Verbindungsart dann gleich wie eine normale Client-Server Verbindung. Dies erlaubt es dem Anbieter nur den Server für das Matchmaking bereit zu stellen. Die Belastung dessen ist wesentlich geringer als bei einer reinen Client-Server Struktur und Latenzen zu diesem Server spielen keine Rolle. So können selbst schwache Server-Infrastrukturen den Anforderungen gerecht werden. Auf der Seite des Spiels hat der Host, der sich selbst als Spieler mit dem lokalen Server verbindet, stehts eine Verzögerung von 0ms. Dies bietet einen gewaltigen Vorteil gegenüber den anderen Clients. Außerdem ist diese Verbindungsart, ähnlich wie reines Peer to Peer, anfällig für Hosts mit schlechter Verbindung. Dazu muss ein eventuelles Beenden des Spiels oder ein Verbindungsabbruch seitens des gewählten Hosts berücksichtigt und entsprechende Ausnahmebehandlung programmiert werden. Üblicherweise bekannt ist diese als „Host Migration“: Dabei wird ein neuer Host bestimmt, der die Aufgaben des alten übernimmt. Anschließend werden die Spielstände erneut synchronisiert und das Spiel wird fortgesetzt. Dies führt zu teils sekundenlangen Unterbrechungen. Durch diese Schwierigkeiten in der Verwaltung der Hostverbindung und den damit einhergehenden, möglichen Qualitätsverlusten, ist das unechte Peer to Peer heute in nur wenigen Spielen gebräuchlich. Ein namentliches Beispiel dafür ist „For Honor“ von Ubisoft Montreal, dessen Verwendung dieser Verbindungsart immer wieder für Kontroversen gesorgt hat.1,2

3 Methodik

Im letzten Kapitel wurden die Voraussetzungen dargelegt die Technologien erfüllen müssen um für die Entwicklung von Netzwerk Komponenten in Frage zu kommen. Im Folgenden werden mögliche Kandidaten vorgestellt und auf ihre Eignung geprüft.

Auswahl der kompatiblen Technologien Darstellung als grafik zeitstrahl wo inkompatible sachen dann runterfallen sozusagen oder eher plus minus rot grün tabelle

3.1 Webtechnologien für TCP-Kommunikation

3.1.1 HTTPS/2

HTTPS/2 ist eine Weiterentwicklung des für das Internet fundamentalen HTTP Protokolls. HTTP dient als einfaches Anfrage-Antwort Protokoll, mithilfe dessen Anfragen an einen Server geschickt und von diesem mit den gewünschten Daten beantwortet werden können. HTTP stellt die Grundlage für die Kommunikation im Internet dar, erlaubt den Austausch von Hypertext Formaten und die Lokalisierung von Ressourcen mithilfe von URLs. HTTP liegt im OSI-Modell (siehe Abbildung 2) auf der siebten, der Anwendungsschicht. HTTP erwartet eine zuverlässige Verbindung und baut als solches auf dem TCP der vierten OSI-Model Schicht auf (siehe Abbildung 2).

HTTP/2 basiert auf dem von Google entwickelten SPDY-Protokoll, mit dem HTTP/2 im Jahr 2015 veröffentlicht wurde. Ziel dieser Weiterentwicklung ist es jedoch nicht das alte Protokoll HTTP/1.1 abzulösen. Durch erweiterte Verhandlungsmechanismen können moderne Browser und Server zwischen den zwei Technologien nach bedarf wechseln und nur mit dem zu Client und Server kompatiblen agieren.

HTTP/2 erreicht im Gegensatz wesentlich bessere Datenraten und Latenzzeiten für Anfragen. Dies wird durch den wechsel von textbasierten Übertragungen, die bei HTTP/1.1 übliche waren, auf binärcode basierte Übertragungen erreicht. Dies vereinfacht das parsen, umwandeln in unterschiedliche Datenformate, der erhaltenen Daten. Zusätzlich komprimiert das HTTP/2 Protokoll die Headerbereiche, wodurch mehrere Bytes pro Anfrage eingespart werden können und die Kombination multipler Header zu nur einem einzelnen möglich ist. Dies spart zusätzlich Datenvolumen ein und erlaubt für schnellere Kommunikation.

Eine weitere, und womöglich die größte, Änderung gegenüber HTTP/1.1 ist die Möglichkeit des Multiplexing.



Abb. 5: Verhalten bei multiplen Anfragen bei Verwendung von HTTP/1.1 oder HTTP/2

Quelle: Das Interwebs https://medium.com/@factoryhr/http-2-the-difference-between-http-1-1-benefits-and-how-to-use-it-38094fa0e95b

Multiplexing bedeutet, dass über eine einzelne Verbindung, meistens TCP, mehrere Anfragen geschickt werden können ohne für jede einzelne Anfrage eine neue Verbindung zu erstellen (siehe Abbildung 5). So ist nur einmal eine Verbindungsverhandlung notwendig, wodurch die Anzahl der gesendeten Header von fünf, zwei seitens des Clients für den Dreiwege Handshake und drei weitere für die jeweiligen Ressourcenanfragen, auf drei reduziert wird. Dies stellt eine Dateneinsparung von 40% dar.

Trotz all dieser Einsparungen ist HTTP/2 dennoch nicht für die Verwendung in Echtzeitspielen geeignet, da auch hier eine zuverlässige und geordnete Abfolge an Packets erwartet wird: Netzwerkstau ist also immernoch ein Problem. Außerdem müssen Verbindungsabbrüche und Wiederverbindungen von Clients explizit gehandhabt werden, da HTTP/2 diese Funktionalitäten nicht nativ mitbringt.

3.1.2 WebSocket

3.1.3 Socket.io

3.2 UDP Kommunikation

3.2.1 Dgram

3.2.2 WebRTC

3.3 Entwicklung

Komponenten

Komp 1

Komp 2

Komp 3

Komp 4

4. Praktische Anwendung

Neue Nachrichten abfangen

Clientenlogik einbauen

Authoritative Commands strukturieren

Ids für netzwerkobjete vergeben

5. Diskussion und Ausblick

Im Rahmen dieser Bachelorarbeit wurden funktionierende Netzwerkkomponenten für die Game-Engine Fudge konzipiert und implementiert. Das Ergebnis zeigt, dass entsprechend der Spezifikationen von WebRTC und Chromium, UDP Kommunikation direkt zwischen Peers möglich ist, und sogar eine Serverstruktur aufgebaut und in Electron implementiert werden kann. Die Verwendung von TypeScript hat die Fehlersuche stark vereinfacht und es erlaubt, einfache Strukturen zu erstellen, die Skalierbarkeit, Wartbarkeit und Modularität gewährleisten Dies zeigt auch eine gewisse Flexibilität in der Herangehensweise. Die hier erstellten Komponenten könnten anderweitig umgesetzt werden, auch wenn die Kommunikation mit dem UDP-Protokoll in einer Browserumgebung ausschließlich über WebRTC erfolgen kann.

In dieser Arbeit wurde dabei die grundlegende Erstellung von Kommunikationswegen über UDP zwischen zwei Clienten, seien beide Peers oder sei es eine Client-Server Struktur, fokussiert. Die Kombination von Webtechnologien mit Servertechnologien im Rahmen einer Browserumgebung hat aufwendige Fehlersuche notwendig gemacht. Die Kommunikation über UDP innerhalb einer modernen Browserumgebung ist aus Sicherheitsgründen eigentlich unmöglich gemacht. Dadurch sind entgegen der Erwartungen Algorithmen zur Serverseitigen-Prediction, dem Austausch von Gamestates und weitläufigen Lobbysystemen im Rahmen der Arbeit nicht möglich gewesen.

Gleichzeitig bieten sich die Komponenten für weiterführende Arbeiten an, beispielsweise um serverseitige Prediction zu konzipieren und umzusetzen, Lobbysysteme zu entwickeln und serverseitige Gamestate Validierung zu implementieren. Zudem können potentiell spezialisierte Netzwerkprotokolle entwickelt werden, die genau auf die didaktischen Anforderungen von Fudge zugeschnitten sind und so die Abhängigkeit von UDP umgehen. Außerdem können optimierte Kommunikationswege erarbeitet werden, die die Latenz bei der Kommunikation zwischen Clients und Server weiter reduziert.

Die für diese Arbeit verwendeten Technologien sind weltweit anerkannt und viel genutzt, beispielsweise JavaScript und Chromium, beide unterstützt durch den Technologiekonzern Google, TypeScript entwickelt von Microsoft, und Electron dass auf Technologie von Microsoft basiert. So besteht für Fudge keine Gefahr in näherer Zukunft obsolete Webtechnologien oder Komponenten deren Kompatibilität nicht mehr gegeben ist zu beherbergen.

Wie können die komponenten selbst weiterentwickelt werden z.b, durch

**server prediction,**

**lazy loading,**

**nachrichten validierung,**

**werte validierung**